Monte Carlo tree search

From Wikipedia, the free encyclopedia

In computer science, **Monte Carlo tree search** (**MCTS**) is a heuristic search algorithm of making decisions in some decision processes, most notably employed in game playing. The leading example of its use is in contemporary computer Go programs,[1] but it is also used in other board games, as well as real-time video games and non-deterministic games such as poker (see history section).

Contents

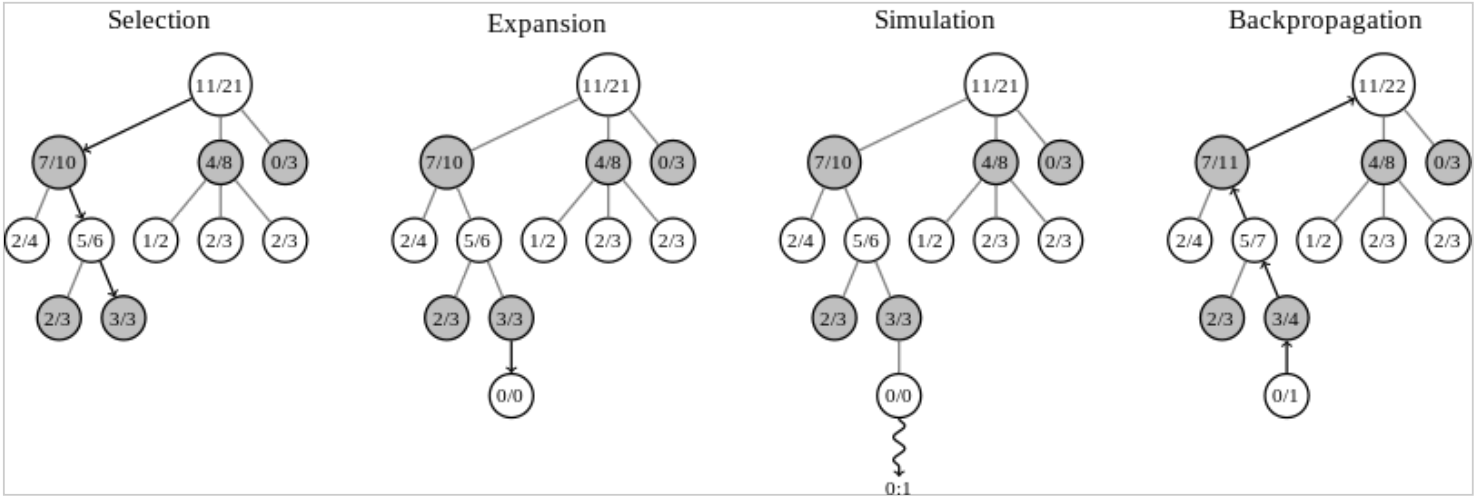
Principle of operation

MCTS concentrates on analysing the most promising moves, basing the expansion of the search tree on random sampling of the search space. MCTS in games is based on many *playouts*. In each playout, the games are played-out to the very end by selecting moves at random. The final game result of each playout is then used to weight the nodes in the game tree so that better nodes are more likely to be chosen in future playouts.

The most primitive way of using playouts is playing the same number of them after each legal move of the current player and choosing the move, after which the most playouts ended up in the player's victory.[2] The efficiency of this method—called *Pure Monte Carlo Game Search*—often increases when, as time goes on, more playouts are assigned to the moves that have frequently resulted in the player's victory (in previous playouts). Full MCTS consists in employing this principle recursively on many depths of the game tree. Each round of MCTS consists of four steps:[3]

- *Selection*: starting from root R, select successive child nodes down to a leaf node L. The section below says more about a way of choosing child nodes that lets the game tree expand towards most promising moves, which is the essence of MCTS.
- *Expansion*: unless L ends the game, either create one or more child nodes of it and choose from them node C.
- *Simulation*: play a random playout from node C.
- *Backpropagation*: using the result of the playout, update information in the nodes on the path from C to R.

Sample steps of one round are shown in the figure below. Each tree node stores the number of won/played playouts.



Steps of Monte Carlo tree search

Note that the updating of the number of wins in each node during backpropagation should be from the point of view of the player who made the move that resulted in that node [4] (this is not accurately reflected in the sample image above). This ensures that during selection, each player chooses to expand towards the most promising moves for that player, which mirrors the maximizing behavior of each player in reality.

Such rounds are repeated as long as the time allotted to a move is not up. Then, one of moves from the root of the tree is chosen but it is the move with the most simulations made rather than the move with the highest average win rate.

Pure Monte Carlo game search

This basic procedure can be applied in all games which have only positions with finitely many moves, and which allow only for games of finite length. In a position all feasible moves are determined, for each one *k* random games are played to the very end, and the cumulative scores are recorded. The move with the best score is chosen. Ties are broken by fair coin flips. Pure Monte Carlo Game Search results in strong play in several games with random elements, for instance in EinStein würfelt nicht!. It converges to optimal play (as *k* tends to infinity) in board filling games with random turn order, for instance in Hex with random turn order.

Exploration and exploitation

The main difficulty in selecting child nodes is maintaining some balance between the *exploitation* of deep variants after moves with high average win rate and the *exploration* of moves with few simulations. The first formula for balancing exploitation and exploration in games, called UCT (*Upper Confidence Bound* 1 *applied to trees*), was introduced by L. Kocsis and Cs. Szepesvári[5] basing on the UCB1 formula derived by Auer, Cesa-Bianchi, and Fischer.[6] Kocsis and Szepesvári recommend to choose in each node of the game tree the move, for which the expression $\frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$ has the highest value. In this formula:

- w_i stands for the number of wins after the ith move;
- n_i stands for the number of simulations after the ith move;
- c is the exploration parameter—theoretically equal to $\sqrt{2}$; in practice usually chosen empirically;
- t stands for the total number of simulations, equal to the sum of all n_i.

The first component of the formula above corresponds to exploitation; it is high for moves with high average win ratio. The second component corresponds to exploration; it is high for moves with few simulations.

Most contemporary implementations of MCTS are based on some variant of UCT.

Advantages and disadvantages

Although it has been proven that the evaluation of moves in MCTS converges to the minimax evaluation,[7] the basic version of MCTS can converge to it after enormous time. Besides this disadvantage (partially cancelled by the improvements described below), MCTS has some advantages compared to alpha–beta pruning and similar algorithms.

Unlike them, MCTS works without an explicit evaluation function. It is enough to implement game mechanics, i.e. the generating of allowed moves in a given position and the game-end conditions. Thanks to this, MCTS can be applied in games without a developed theory or even in general game playing.

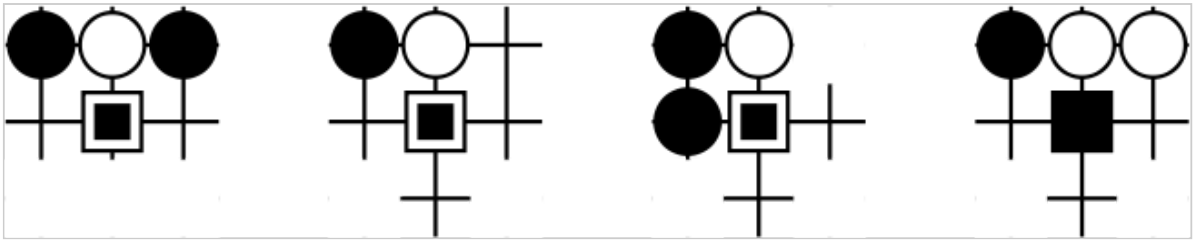
The game tree in MCTS grows asymmetrically: the method concentrates on searching its more promising parts. Thanks to this, it achieves better results than classical algorithms in games with a high branching factor.

Moreover, MCTS can be interrupted at any time, yielding the move it considers the most promising.

Improvements

Various modifications of the basic MCTS method have been proposed, with the aim of shortening the time to find good moves. They can be divided into improvements based on expert knowledge and into domain-independent improvements.

MCTS can use either *light* or *heavy* playouts. Light playouts consist of random moves. In heavy playouts various heuristics influence the choice of moves.[8] The heuristics can be based on the results of previous playouts (e.g. the Last Good Reply heuristic[9]) or on expert knowledge of a given game. For instance, in many go-playing programs, certain stone patterns on a part of the board influence the probability of moving into that part.[10] Paradoxically, not always playing stronger in simulations makes an MCTS program play stronger overall.[11]



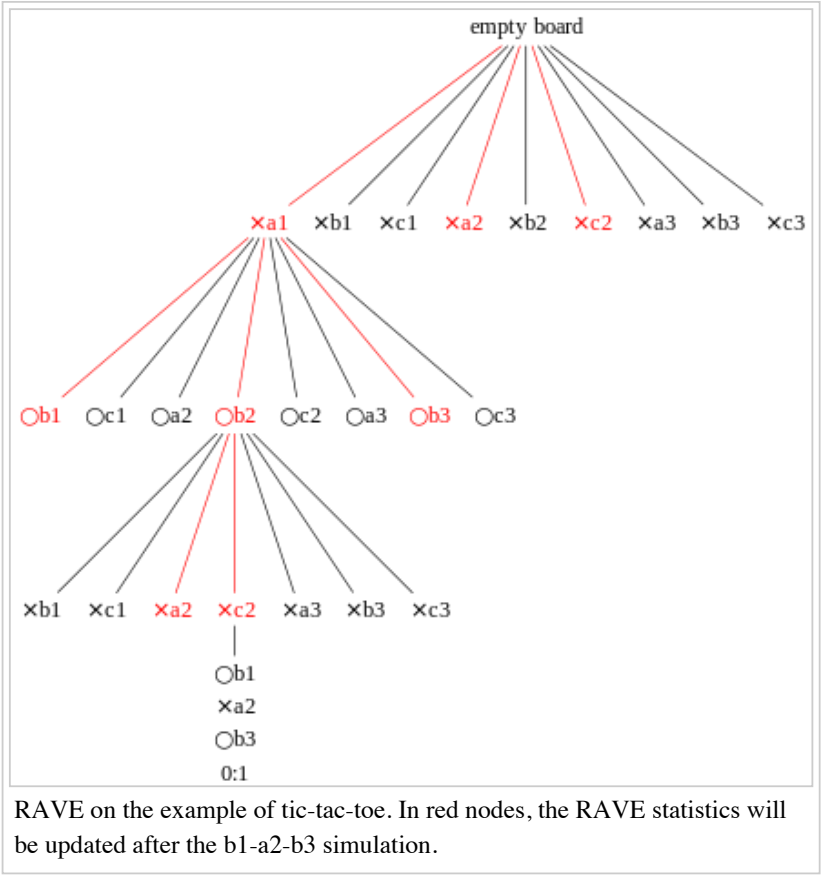
Patterns of *hane* (surrounding opponent stones) used in playouts by the MoGo program. It is advantageous both for black and for white to put a stone on the middle square, except the rightmost pattern where it is advantageous for black only.[10]

Domain-specific knowledge can also be used while building the game tree, to help the exploitation of some variants. One of such methods consists in assigning nonzero *priors* to the numbers of won and played simulations when creating a child node. Such artificially raised or lowered average win rate causes the node to be chosen more or less frequently, respectively, in the selection step.[12] A related method, called *progressive bias*, consists in adding to the UCB1 formula a $\frac{b_i}{n_i}$ element, where b_i is a heuristic score of the ith move.[3]

The basic MCTS collects enough information to find the most promising moves after many rounds. Before that, it chooses more or less random moves. This initial phase can be significantly shortened in a certain class of games thanks to RAVE (*Rapid Action Value Estimation*).[12] The games in question are those where permutations of a sequence of moves lead to the same position, especially board games where a move consists in putting a piece or a stone on the board. In such games, the value of a move is often only slightly influenced by moves played elsewhere.

In RAVE, for a given game tree node N, its child nodes C_i store besides the statistics of wins in playouts started in node N also the statistics of wins in all playouts started in node N and below it, if they contain move i (also when the move was played in the tree, between node N and a playout). This way, the contents of tree nodes are influenced not only by moves

played immediately in a given position but also by the same moves played later.



RAVE on the example of tic-tac-toe. In red nodes, the RAVE statistics will be updated after the b1-a2-b3 simulation.

When using RAVE, the selection step selects the node, for which the modified UCB1 formula $(1-\beta(n_i,\tilde{n}_i))\frac{w_i}{n_i}+\beta(n_i,\tilde{n}_i)\frac{\tilde{w}_i}{\tilde{n}_i}+c\sqrt{\frac{\ln t}{n_i}}$ has the highest value. In this formula, $\tilde{w}_i$ and $\tilde{n}_i$ stand for the number of won playouts containing move i and the number of all playouts containing move i, and the $\beta(n_i,\tilde{n}_i)$ function should be close to one and to zero for relatively small and relatively big n_i and $\tilde{n}_i$, respectively. One of many formulas for $\beta(n_i,\tilde{n}_i)$, proposed by D. Silver,[13] says that in balanced positions one can take $\beta(n_i,\tilde{n}_i)=\frac{\tilde{n}_i}{n_i+\tilde{n}_i+4b^2n_i\tilde{n}_i}$, where b is an empirically chosen constant.

Heuristics used in MCTS often depend on many parameters. There are methods of automatic tuning the values of these parameters to maximize the win rate.[14]

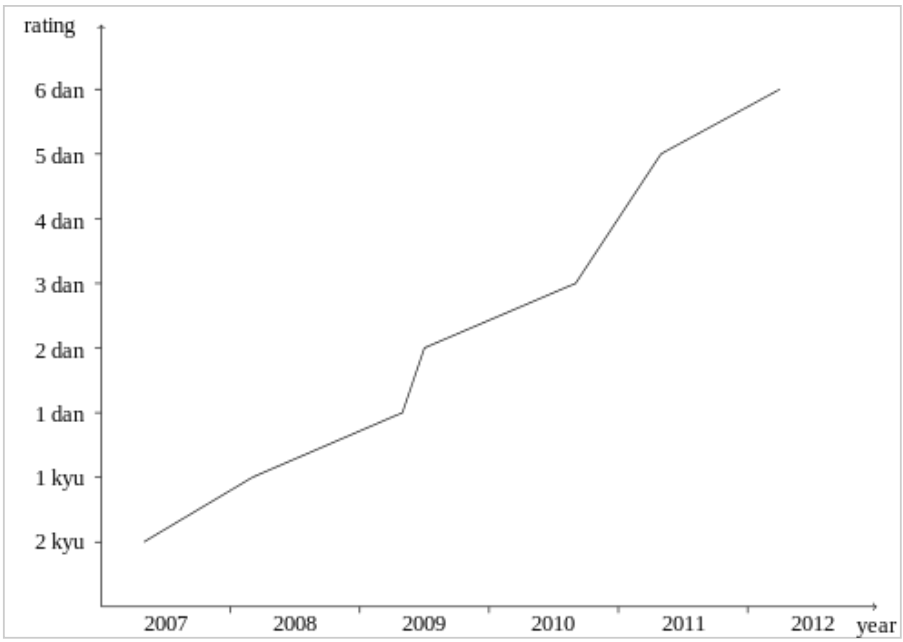
MCTS can be concurrently executed by many threads or processes. There are several fundamentally different methods of its parallel execution:[15]

- *Leaf parallelization*, i.e. parallel execution of many playouts from one leaf of the game tree.
- *Root parallelization*, i.e. building independent game trees in parallel and making the move basing on the root-level branches of all these trees.
- *Tree parallelization*, i.e. parallel building of the same game tree, protecting data from simultaneous writes either with one, global mutex, with more mutexes, or with non-blocking synchronization.[16]

History

The Monte Carlo method, based on random sampling, dates back to the 1940s. Bruce Abramson explored the idea in his 1987 PhD thesis and said it "is shown to be precise, accurate, easily estimable, efficiently calculable, and domain-independent."[17] He experimented in-depth with Tic-tac-toe and then with machine generated evaluation functions for Othello and Chess. In

1992, B. Brügmann employed it for the first time in a go-playing program,[2] but his idea was not taken seriously. In 2006, called the year of the Monte Carlo revolution in Go,[18] R. Coulom described the application of the Monte Carlo method to game-tree search and coined the name Monte Carlo tree search,[19] L. Kocsis and Cs. Szepesvári developed the UCT algorithm,[5] and S. Gelly et al. implemented UCT in their program MoGo.[10] In 2008, MoGo achieved dan (master) level in 9×9 go,[20] and the Fuego program began to win with strong amateur players in 9×9 go.[21] In January 2012, the Zen program won 3:1 a 19×19 go match with John Tromp, a 2 dan player.[22]



The rating of best go-playing programs on the KGS server since 2007. Since 2006, all the best programs use MCTS.[23]

MCTS has also been used in programs that play other board games (for example Hex,[24] Havannah,[25] Game of the Amazons,[26] and Arimaa[27]), real-time video games (for instance Ms. Pac-Man,[28][29] Fable Legends[30] and Total War: Rome II[31]), and nondeterministic games (such as skat,[32] poker,[33] Magic: The Gathering,[34] or Settlers of Catan[35]).

References

1. "MCTS.ai: Everything Monte Carlo Tree Search". Retrieved 2012-02-19.
2. Brügmann, Bernd (1993). *Monte Carlo Go* (PDF). Technical report, Department of Physics, Syracuse University.
3. G.M.J.B. Chaslot, M.H.M. Winands, J.W.H.M. Uiterwijk, H.J. van den Herik, B. Bouzy (2008). "Progressive Strategies for Monte-Carlo Tree Search" (PDF). *New Mathematics and Natural Computation* **4** (3): 343–359. doi:10.1142/s1793005708001094.
4. http://mcts.ai/code/python.html
5. Kocsis, Levente; Szepesvári, Csaba (2006). "Bandit based Monte-Carlo Planning". *Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18–22, 2006, Proceedings. Lecture Notes in Computer Science 4212*. Johannes Fürnkranz, Tobias Scheffer, Myra Spiliopoulou (eds.). Springer. pp. 282–293. ISBN 3-540-45375-X.
6. Auer, Peter; Cesa-Bianchi, Nicolò; Fischer, Paul (2002). "Finite-time Analysis of the Multiarmed Bandit Problem" (PDF). *Machine Learning* **47**: 235–256. doi:10.1023/a:1013689704352.
7. Bouzy, Bruno. "Old-fashioned Computer Go vs Monte-Carlo Go". *IEEE Symposium on Computational Intelligence and Games, April 1–5, 2007, Hilton Hawaiian Village, Honolulu, Hawaii* (PDF).
8. Swiechowski, M.; Mandziuk, J., "Self-Adaptation of Playing Strategies in General Game Playing" (2010), *IEEE Transactions on Computational Intelligence and AI in Games*, doi: 10.1109/TCIAIG.2013.2275163, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6571225&isnumber=4804729
9. Drake, Peter (December 2009). "The Last-Good-Reply Policy for Monte-Carlo Go". *ICGA Journal* **32** (4): 221–227.
10. Sylvain Gelly, Yizao Wang, Rémi Munos, Olivier Teytaud (November 2006). *Modification of UCT with Patterns in Monte-Carlo Go* (PDF). Technical report, INRIA.

11. Seth Pellegrino, Peter Drake (2010). "Investigating the Effects of Playout Strength in Monte-Carlo Go". *Proceedings of the 2010 International Conference on Artificial Intelligence, ICAI 2010, July 12–15, 2010, Las Vegas Nevada, USA*. Hamid R. Arabnia, David de la Fuente, Elena B. Kozerenko, José Angel Olivas, Rui Chang, Peter M. LaMonica, Raymond A. Liuzzi, Ashu M. G. Solo (eds.). CSREA Press. pp. 1015–1018. ISBN 1-60132-148-1.
12. Sylvain Gelly, David Silver (2007). "Combining Online and Offline Knowledge in UCT". *Machine Learning, Proceedings of the Twenty-Fourth International Conference (ICML 2007), Corvallis, Oregon, USA, June 20–24, 2007* (PDF). Zoubin Ghahramani (ed.). ACM. pp. 273–280. ISBN 978-1-59593-793-3.
13. David Silver (2009). *Reinforcement Learning and Simulation-Based Search in Computer Go* (PDF). PhD thesis, University of Alberta.
14. Rémi Coulom. "CLOP: Confident Local Optimization for Noisy Black-Box Parameter Tuning". *ACG 2011: Advances in Computer Games 13 Conference, Tilburg, the Netherlands, November 20–22*.
15. Guillaume M.J-B. Chaslot, Mark H.M. Winands, H. Jaap van den Herik (2008). "Parallel Monte-Carlo Tree Search". *Computers and Games, 6th International Conference, CG 2008, Beijing, China, September 29 – October 1, 2008. Proceedings* (PDF). H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, Mark H. M. Winands (eds.). Springer. pp. 60–71. ISBN 978-3-540-87607-6.
16. Markus Enzenberger, Martin Müller (2010). "A Lock-free Multithreaded Monte-Carlo Tree Search Algorithm". *Advances in Computer Games: 12th International Conference, ACG 2009, Pamplona, Spain, May 11–13, 2009, Revised Papers*. H. Jaap Van Den Herik, Pieter Spronck (eds.). Springer. pp. 14–20. ISBN 978-3-642-12992-6.
17. Abramson, Bruce (1987). *The Expected-Outcome Model of Two-Player Games* (PDF). Technical report, Department of Computer Science, Columbia University. Retrieved 23 December 2013.
18. Rémi Coulom (2008). "The Monte-Carlo Revolution in Go". *Japanese-French Frontiers of Science Symposium* (PDF).
19. Rémi Coulom (2007). "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search". *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29–31, 2006. Revised Papers*. H. Jaap van den Herik, Paolo Ciancarini, H. H. L. M. Donkers (eds.). Springer. pp. 72–83. ISBN 978-3-540-75537-1.
20. Chang-Shing Lee, Mei-Hui Wang, Guillaume Chaslot, Jean-Baptiste Hoock, Arpad Rimmel, Olivier Teytaud, Shang-Rong Tsai, Shun-Chin Hsu, Tzung-Pei Hong (2009). "The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments" (PDF). *IEEE Transactions on Computational Intelligence and AI in Games* **1** (1): 73–89. doi:10.1109/tciaig.2009.2018703.
21. Markus Enzenberger, Martin Mūller (2008). *Fuego – An Open-Source Framework for Board Games and Go Engine Based on Monte Carlo Tree Search* (PDF). Technical report, University of Alberta.
22. "The Shodan Go Bet". Retrieved 2012-05-02.
23. "Sensei's Library: KGSBotRatings". Retrieved 2012-05-03.
24. Broderick Arneson, Ryan Hayward, Philip Henderson (June 2009). "MoHex Wins Hex Tournament" (PDF). *ICGA Journal* **32** (2): 114–116.
25. Timo Ewalds (2011). *Playing and Solving Havannah* (PDF). Master's thesis, University of Alberta.
26. Richard J. Lorentz (2008). "Amazons Discover Monte-Carlo". *Computers and Games, 6th International Conference, CG 2008, Beijing, China, September 29 – October 1, 2008. Proceedings*. H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, Mark H. M. Winands (eds.). Springer. pp. 13–24. ISBN 978-3-540-87607-6.
27. Tomáš Kozelek (2009). *Methods of MCTS and the game Arimaa* (PDF). Master's thesis, Charles University in Prague.
28. Xiaocong Gan, Yun Bao, Zhangang Han (December 2011). "Real-Time Search Method in Nondeterministic Game – Ms. Pac-Man". *ICGA Journal* **34** (4): 209–222.
29. Tom Pepels, Mark H. M. Winands, Marc Lanctot (September 2014). "Real-Time Monte Carlo Tree Search in Ms Pac-Man". *IEEE Transactions on Computational Intelligence and AI in Games* **6** (3): 245–257. doi:10.1109/tciaig.2013.2291577.
30. "Tactical Planning and Real-time MCTS in Fable Legends". Retrieved 2015-08-27.
31. "Monte-Carlo Tree Search in TOTAL WAR: ROME II's Campaign AI". Retrieved 2014-08-13.
32. Michael Buro, Jeffrey Richard Long, Timothy Furtak, Nathan R. Sturtevant (2009). "Improving State Evaluation, Inference, and Search in Trick-Based Card Games". *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11–17, 2009*. Craig Boutilier (ed.). pp. 1407–1413.
33. Jonathan Rubin, Ian Watson (April 2011). "Computer poker: A review" (PDF). *Artificial Intelligence* **175** (5–6): 958–987. doi:10.1016/j.artint.2010.12.005.
34. C.D. Ward, P.I. Cowling (2009). "Monte Carlo Search Applied to Card Selection in Magic: The Gathering". *CIG'09 Proceedings of the 5th international conference on Computational Intelligence and Games* (PDF). IEEE Press.
35. István Szita, Guillaume Chaslot, Pieter Spronck (2010). "Monte-Carlo Tree Search in Settlers of Catan". *Advances in Computer Games, 12th International Conference, ACG 2009, Pamplona, Spain, May 11–13, 2009. Revised Papers* (PDF). H. Jaap van den Herik, Pieter Spronck (eds.). Springer. pp. 21–32. ISBN 978-3-642-12992-6.

Bibliography

- Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, Simon Colton (March 2012). "A Survey of Monte Carlo Tree Search Methods" (PDF). *IEEE Transactions on Computational Intelligence and AI in Games* **4** (1).

Categories: Combinatorial game theory | Heuristic algorithms | Monte Carlo methods